DMBench: Load Testing and Benchmarking Tool for Data Migration

Fares Hamouda
York University
North York, Canada
faresham@yorku.ca

Marios Fokaefs
York University
North York, Canada
fokaefs@yorku.ca

Dariusz Jania
IBM
Kraków, Poland
dariusz.jania@pl.ibm.com

ABSTRACT

Data migration refers to the set of tasks around transferring data over a network between two systems, either homogeneous or heterogeneous, and the potential reformatting of this data. Combined with large volumes of data, resource constraints and variety in data models and formats, data migration can be critical for enterprises, as it can consume a significant amount of time, incur high costs, and pose a significant risk if not executed correctly. The ability to accurately and effectively predict these challenges and plan for proper resource, time and budget allocation is vital for the proper execution of data migration. In this work, we introduce the concept of load testing and benchmarking for data migration to allow decision-makers for higher efficiency and effectiveness when planning for such tasks. Our framework aims for extensibility and customizability to enable the execution of a greater variety of tests. Here, we present a prototype architecture, a roadmap of how the development of such a platform should proceed and a simple case study of how it can be used in practice.

CCS CONCEPTS

• **General and reference** → **Experimentation**; • **Computer systems organization** → **Cloud computing**; • **Information systems** → **Cloud based storage**; **Database performance evaluation**.

KEYWORDS

data migration; big data; benchmarking; load testing; software performance; data integrity; data transfer

ACM Reference Format:
Fares Hamouda, Marios Fokaefs, and Dariusz Jania. 2024. DMBench: Load Testing and Benchmarking Tool for Data Migration. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '24), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3653663

1 INTRODUCTION

Technological advancements often drive large-scale migrations of software and data systems to new platforms, presenting challenges in data transfer. The volume of enterprise data, ranging from terabytes to petabytes, strains networks and increases the risk of errors. Additionally, diverse target systems may necessitate changes in migration strategies. In a business context, data migration is viewed as a maintenance task that requires resources and careful planning to minimize disruption to regular business operations.

To tackle the planning and resource challenges, our prototype architecture presents a flexible testing platform. It enables swift and adaptable experimentation to systematically assess various migration scenarios and configurations, yielding comprehensive data for informed decisions. Adding to its value, this framework extends its utility to the execution of migration experiments by simulating multiple production workloads on the data source machine. This approach enables a comprehensive assessment of how the workload on the data source machine influences the performance and reliability of the data migration process. Additionally, the framework allows for an investigation into the reciprocal impact—how the migration process affects production workloads and performance on the data source. We maintain that this prototype's inherent flexibility can readily accommodate such complex scenarios, positioning it not only as a benchmarking tool but also as a versatile resource for in-depth migration experimentation and analysis.

In this work, we outline the main components of the prototype architecture while highlighting essential non-functional requirements crucial for platform design. We assert that the prototype's inherent flexibility allows for the accommodation of more complex migration scenarios and configurations. Furthermore, we envision this platform evolving beyond its benchmarking capabilities to serve as a dynamic tool for data collection, decision-making, or even as the foundation for a dynamically adaptive migration strategy.

2 RELATED WORK

Data migration has mainly been studied in the context of migration to the cloud [6, 9, 15]. Some studies investigate methods for validating data migration to ensure data accuracy post-migration. These approaches include comparing data sets migrated using different protocols, validating migration based on comparison data, and employing techniques such as checksums or key-data pairs [7, 10, 11, 16]. Additionally, research on optimizing the migration process focuses on minimizing testing costs, customizing migration problems with specific constraints, and proposing streamlined algorithms [2, 4, 5, 8]. Besides practitioners, our platform can also support this research by allowing testing of novel validation and optimization techniques. Subramani et al. [13] propose a theoretical algorithm to optimize data migration in order to minimize

ICPE Companion '24 , May 7–11, 2024, London, United Kingdom

ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3653663

testing on the application side. However, they do not discuss any method or tool for testing the actual migration.

3 DMBENCH ARCHITECTURE AND IMPLEMENTATION

DMBench is designed to be primarily functional, usable and portable, but also flexible and extensible. To this end, it consists of multiple semi-independent modules with distinct roles that can be easily configured individually or as a whole, and can be replaced by alternative implementations with ease. The use of Docker containers makes the platform easy to deploy and redeploy at will and on demand. Next, we provide more details about the properties and relations between these modules.

3.1 Functional and Non-Functional Requirements

The primary objective of DMBench is to allow testers and decision-makers to design and execute migration experiments as efficiently and as effectively as possible. Such a platform should allow for easy configuration of different experiments, fast deployment and execution of the experiment and fully automated and comprehensive presentation of the results. In principle, the platform should enable testers to execute as many "what-if" scenarios as possible to compare many alternatives or confirm many hypotheses. In practice, the platform guides the tester to prepare an environment to send data from one machine to another (potentially remote) with different types of configurations, and then monitors each part of the process.

DMBench has been designed according to a set of principles on the functional and non-functional requirements of a data migration testing tool, as described below.

Functional Requirements:

- **Easy setup:** The setup for a migration task refers to all steps relevant to preparing the source and target machines, the migration engine, any metering apparatus (e.g., logging or monitoring) along with respective databases to save results, and the controller of the migration test or experiment. Setup can be a time-consuming step, especially when it is not automated. The ability to quickly complete the setup and make it easy to share between experiments is vital to the ability of testing multiple scenarios and configurations.
- **Easy configuration:** Numerous parts of the platform and of an experiment need to be configurable to allow for extensive control to the tester. If the configuration is complex, inconsistent or generally lacking, it may render experimentation results unusable.
- **Convenient and complete access to results:** For any benchmark to be useful, it needs to provide the results in a convenient way to enable further analysis and interpretation. While this can include reports with visualizations and descriptive statistics, it is important to also return all raw measurements and results in a format that can be easily digested by an analysis software.

Non-functional Requirements.

- **Usability:** The user of the tool is expected to have some basic understanding of data migration and potentially of the source and target systems, and the data to be transferred. Besides that, the tools should hide as many details about the experimentation infrastructure as possible and expose any configuration or input points through a clear and easy-to-use interface. The proposed benchmark follows a simplified approach, allowing users to initiate experiments with a few straightforward commands after the environment is set up and configured. The objective is to streamline the process, ensuring ease of use while maintaining simplicity.
- **Extensibility:** The benchmark as a software framework. Through the configuration files, the user can provide outside input to the framework and provide any migration engine she wants to test by dockerizing it following our process. In addition, all other modules, including logging and monitoring, can be replaced by what the user desires, as long as they can be deployed on Docker. Besides, the "frozen spots" of the framework, i.e., the abstract components or the components that the user cannot override, dictate the flow of the experiment.
- **Accuracy and Reliability:** When configuring an experiment, the user has the possibility to request for each experiment to be executed a given number of times. By repeating the exact same experiment multiple times and averaging over the results of the iterations, we can ensure that any source of variability is excluded, and the results are accurate and reliable. In many analyses, it is required to perform statistical testing to confirm or reject our initial hypothesis. By repeating the experiments multiple times, we make it possible to run such tests with high confidence. No matter the number of repetitions, experiments in DMBench are executed deterministically and any variations originates from the environment or the use case and not from the tool. Through complete access to the results, this is verifiable by the tester.

3.2 Architecture

Figure 1 shows the main components of DMBench, which are described in detail next.

- **Migration Engine:** Central to the framework, the Migration Engine serves as the focal point for all bench-marking activities, but is primarily the module responsible for transferring data from the source to the target. All other components within the framework are designed to closely monitor and assess the performance of the engine. Users are afforded the flexibility to select any migration engine of their preference for bench-marking purposes. Whether the chosen engine operates within a singular service architecture or spans multiple services, the only prerequisite is the dockerization of the selected engine. The framework seamlessly manages the intricacies of the bench-marking process, offering a streamlined and user-friendly experience.
- **Data Source:** This component serves as the starting point for the Migration Engine to transfer data from here to a designated target machine. DMBench requires only an IP address and appropriate credentials to connect to the Data

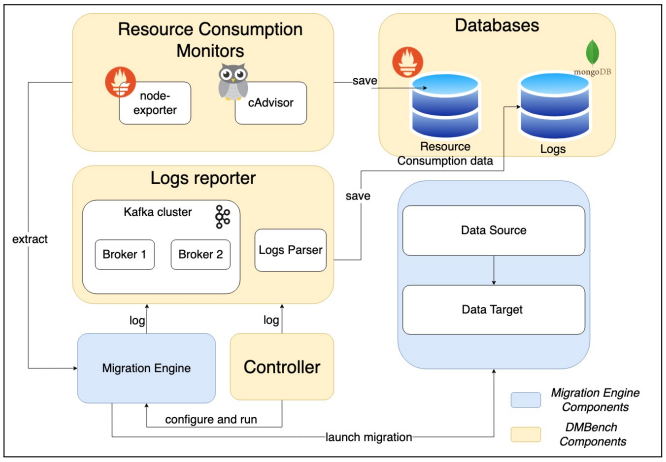

Figure 1: The general architecture of DMBench.

Source machine, but the specific deployment method (virtual machine, physical server, container) is irrelevant to the platform.

- **Data Target:** This component is the destination for data transferred by the Migration Engine from the source machine. Similar to the sources, only an IP address and connection credentials are necessary.
- **Controller:** Responsible for orchestrating all experiments under consistent conditions, the Controller configures the migration engine for each experiment with varying parameters. It initiates and oversees the execution of the migration process, monitoring the engine's performance through recording and analysis of migration logs. Simultaneously, the Controller tracks resource consumption by deploying `cAdvisor` [14]and `node-exporter` [12] on the migration infrastructure.
- **Metrics & Logs Databases:** DMBench uses two databases for the monitored data. The first, a time series database based on `Prometheus` [1], aggregates resource consumption data collected from `cAdvisor` [14]. The second database, a `MongoDB` [3] instance, serves as the repository for all log data generated during the experiments. These databases work in tandem, with `Prometheus` [1] focusing on resource metrics and the second database storing all logs from both the framework and the migration engine, it ensures comprehensive and organized storage of experiment results.
- **Logs reporter:** Comprising two integral components, the Logs Reporter ensures a robust and organized handling of experiment logs. The first component involves a `Kafka` cluster, serving as the repository for all logs. Both the Controller and the Migration Engine publish their logs to `Kafka`, with a dedicated consumer responsible for retrieving and temporarily storing these logs in local files.
  The second component is the parser, which not only extracts data from the logs but also transforms it into a human-readable format. The parsed information is then exported into CSV files before being permanently stored in a `NoSQL` database, as mentioned above. This dual-component approach ensures a seamless and efficient process for managing, interpreting, and extracting insights from the experiment logs.

In our framework, the configuration is specified in a config.ini format, which is parsed within the framework. The framework ensures that the config.ini file is available in a specified path within the Docker container where the migration engine is deployed. The engine then reads the config.ini file to run the experiment based on the provided values. While this approach has been tested with our engine and the DB2 migration engine using a simple Python script to read and execute the configuration, it's important to highlight that the specifics of handling the configuration may vary depending on the engine being used. Therefore, users are responsible for implementing the necessary scripts or procedures to ensure compatibility with their chosen engine. For detailed config.ini explanations, including section meanings and purposes, visit the GitHub repository[1], these details are found in the controller configuration section.

4 CURRENT IMPLEMENTATION STATE

DMBench can presently accommodate: a) a proprietary **default migration engine**, designed for migrating files from a source to a target machine, b) a **MySQL database migration engine**, where the database is first dumped into a file and subsequently migrated using our default migration engine, and c) the **IBM DB2 migration engine**, which is discussed below. To facilitate the adoption of DMBench, comprehensive guidelines on its utilization and further development are meticulously documented and available in a GitHub repository[2]. These guidelines provide users with step-by-step instructions and best practices, ensuring a smooth and efficient migration process.

In addition to the various migration engines, the framework supports a number of different migration scenarios in terms of size and format of data: a) Exploring multiple compression techniques (e.g., GZIP, LZ4) or migrating data over multiple streams, to optimize migration engine configurations. b)The framework enables the assessment of migration engine limitations, such as maximum stream capacity or data volume, by incrementally increasing parameters until system constraints are reached. c) A key feature of the framework is its capability to compare multiple migration engines using predefined datasets, allowing for a comparative analysis of their performance characteristics during the migration process.

4.1 Case Study: IBM DB2 Migration

One practical use case in our framework focuses on migrating data between two IBM Db2 databases. This migration is facilitated by IBM's dedicated migration service, initially containerized using Docker. Following the guidelines detailed in the documentation of our GitHub repository, we establish the required environment. Furthermore, the Controller is configured based on the specifications provided in the documentation.

(1) Ensuring the requisite machines are set up and accessible, we've verified the readiness of both the source and target machines, which are IBM Db2 databases.

[1]https://github.com/yorku-ease/DataMigrationBenchmarkingTool?tab=readme-ov-file#configuration
[2]https://github.com/yorku-ease/DataMigrationBenchmarkingTool

(2) In the configuration phase, we focused on configuring various components of the framework. Notably, the configuration of the Controller is detailed in the Table 1.
(3) Subsequently, the experiment was executed following the steps outlined in the documentation, and the results were obtained and stored in both MongoDB [3] and Prometheus [1].

Key	Value
Section : targetServer	
host	192.168.122.52
username	db2inst1
password	password
port	50000
type	db2
Section : sourceServer	
host	192.168.122.28
username	db2inst1
password	password
port	50000
Section : KafkaCluster	
host	192.168.122.145
port	9092
performanceBenchmarkTopic	performanceBenchmark
migrationEngineTopicName	migrationEngine
frameworktopicname	framework
Section : migrationEnvironment	
migrationEngineDockerImage	fareshamouda/d-b2migrationservice
loggingId	
numberofexperiments	1
Section : experiment	
compress	NO,GZIP,LZ4
maxStreams	3
sourceDatabasetoTargetDatabase	sample=>testdb
tables	DEPARTMENT

Table 1: Configuration parameters passed to the controller for the IBM Db2 case study.

5 FUTURE DEVELOPMENT

The primary aim of the benchmark is to generate data for decision-making and to enhance understanding of migration tasks and systems. In this context, DMBench will be used to accumulate a significant data and knowledge base on the performance of data migration tools and tasks under a large variety of migration scenarios. This database will then be used to develop performance models, enabling simulations and faster decision-making. In addition, DMBench will be extended to support more migration engines to allow for comparisons and effective choice making for practitioners.

6 CONCLUSION

DMBench is a flexible framework for testing the performance of data migration engines. Through systematic setup, configuration, and execution, the framework proves its adaptability to diverse migration scenarios. Leveraging Docker technology and robust logging, it efficiently captures performance benchmarks and resource consumption data.

REFERENCES

[1] Julius Volz and Björn Rabenstein and Matt Bostock. 2012. *Prometheus : an open-source monitoring and alerting toolkit.* SoundCloud. https://prometheus.io/

[2] Eric Anderson, Joe Hall, Jason Hartline, Michael Hobbs, Anna R. Karlin, Jared Saia, Ram Swaminathan, and John Wilkes. 2001. An Experimental Study of Data Migration Algorithms. In *Algorithm Engineering*, Gerth Stølting Brodal, Daniele Frigioni, and Alberto Marchetti-Spaccamela (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 145–158.

[3] Dwight Merriman, Eliot Horowitz, and Kevin Ryan. 2007. *MongoDB: an open-source, document-oriented NoSQL database.* DoubleClick. https://www.mongodb.com/

[4] M Elamparithi and V Anuratha. 2015. A Review on Database Migration Strategies, Techniques and Tools. *World Journal of Computer Application and Technology* 3, 3 (2015), 41–48.

[5] Zhao JF. and Zhou JT. 2014. Strategies and Methods for Cloud Migration. *International Journal of Automation and Computing* 11 (2014), 143–152.

[6] Kevin Kline, Denis McDowell, Dustin Dorsey, and Matt Gordon. 2022. Moving Your Data to the Cloud. In *Pro Database Migration to Azure: Data Modernization for the Enterprise.* Springer, Berlin, Germany, 263–283.

[7] TN Manjunath, Ravindra S Hegadi, and HS Mohan. 2011. Automated data validation for data migration security. *International Journal of Computer Applications* 30, 6 (2011), 41–46.

[8] Johny Morris. 2012. *Practical data migration.* BCS, The Chartered Institute, London, United Kingdom.

[9] Stephen Orban. 6. Strategies for Migrating Applications to the Cloud. *Medium. Library Catalog: medium. com* 6 (6).

[10] PR Devale P Paygude. 2013. Automated Data Validation Testing Tool for Data Migration Quality Assurance. *International Journal of Modern Engineering Research (IJMER)* 3 (2013), 599–603.

[11] Priyanka Paygude and PR Devale. 2013. Automation of data validation testing for QA in the project of DB migration. *International Journal of Computer Science* 3, 2 (2013), 15–22.

[12] Prometheus community. [n. d.]. *Node Exporter: a software component used in conjunction with Prometheus for monitoring Linux and UNIX system.* https://github.com/prometheus/node_exporter

[13] K. Subramani, Bugra Caskurlu, and Alvaro Velasquez. 2019. Minimization of Testing Costs in Capacity-Constrained Database Migration. In *Algorithmic Aspects of Cloud Computing*, Yann Disser and Vassilios S. Verykios (Eds.). Springer International Publishing, Cham, 1–12.

[14] Google Core Team. 2014. *cAdvisor: an open-source container monitoring and performance analysis tool.* Google. https://github.com/google/cadvisor

[15] Jinesh Varia. 2010. Migrating your existing applications to the aws cloud. *A Phase-driven Approach to Cloud Migration* (2010), 1–23.

[16] Bin Wei and Tennyson X Chen. 2014. *Verifying Data Migration Correctness: The Checksum Principle.* RTI Press, United States.